↑P
↑P↑M↑←

***** ALEX *****

fp-algorithms.text >cm>sprint>f0402 A:

7/27/87 17:22:33

-*- Mode: Text; Package: CMI; Base: 10 -*-

This describes every instructions that so far has been planned
to implement using Sprint and Weitek chip. The following are
listed for every instruction: what kind of instruction,
addressing modes, conditionalization on or off, the alogorithm
for the instruction according to which chip is being used,
the ucode space and the performance.

Exception handling is listed at the end of this file, due
to the fact that a lot of it is common to most routines.

```
Instructions:   F+,F-,F*
Precision:      single
Addr-Mode:      two
Condition:      cond/always

Algorithm on WTL-3132:

A op B -> A     32 bit :conditional or always
#cycles         MB              FB
-------         ----------      ----------
32              B0->Td          free
32              A0->Ta          Td->Reg
32        +---->B1->Td          Ta op Reg->Reg
32        |     A1->Tb          Reg->Tc or Ta->Tc
32        |     Tc->A0          Td->Reg
32        |     B2->Tc          Tb op Reg->Reg
32        |     A2->Ta          Reg->Td or Tb->Td
32        +-----Td->A1          Tc->Reg

Ucode Space:    Common routine: 291
                F+:              13
                F-:              13
                F*:              13

Performance:    NO-VPs          VPs
                165 cycles      100 cycles per VP
                               + 65 cycles for overhead
Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.

*****************************************************************************

Instructions:   F+,F-,F*
Precision:      double
Addr-Mode:      two
Condition:      cond/always

Algorithm on WTL-3132:
Ucode Space:    Common routine:
                F+:
                F-:
                F*:
Performance:    NO-VPs          VPs
Exception Detection Modes:
```

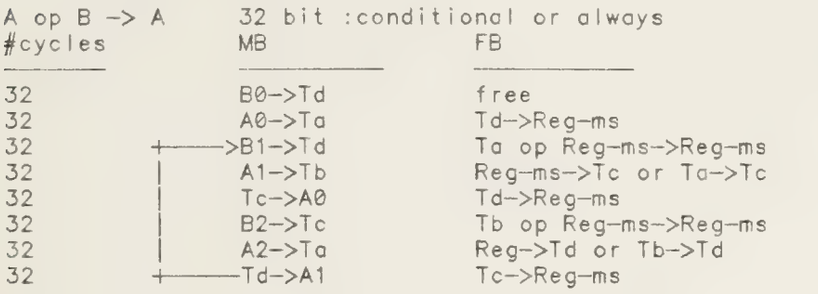
```
Instructions:   F+,F-,F*,F/
Precision:      double
Addr-Mode:      two
Condition:      cond

Algorithm on WTL-3164:

A op B -> A     64 bit :conditional
#cycles         MB              FB
-------         ----------      ----------
32              B0-ls->Ta       free
32              B0-ms->Tb       Ta->Reg-ls
32        +---->A0-ls->Tc       Tb->Reg-ms
32        |     A0-ms->Td       free
32        |     free            Tc,Td op Reg->Reg
32        |     free            Tc,Td op Reg->Reg
32        |     B1-ls->Td       Reg-ls->Ta or Tc->Ta
32        |     Ta->A0-ls       Td->Reg-ls
32        |     B1-ms->Ta       Reg-ms->Tb or Td->Tb
32        +---->Tb->A0-ms       Ta->Reg-ms

Ucode Space:    Common routine: 325
                F+:              13
                F-:              13
                F*:              13
                F/:              25

Performance:    NO-VPs          VPs
                325 cycles      260 cycles per VP
                               + 65 cycles for overhead

F/:             additional 448 cycles

Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.

*************************************************************************

Instructions:   F+,F-,F*,F/
Precision:      double
Addr-Mode:      two
Condition:      always

Algorithm on WTL-3164:

A op B -> A     64 bit :always
#cycles         MB              FB
-------         ----------      ----------
32              B0-ls->Ta       free
32              B0-ms->Tb       Ta->Reg-ls
32              A0-ls->Tc       Tb->Reg-ms
32              A0-ms->Td       free
32        +---->B1-ls->Ta       Tc,Td op Reg->Reg
32        |     B1-ms->Tb       Tc,Td op Reg->Reg
32        |     A1-ls->Tc       Reg-ls->Td
32        |     Td->A0-ls       Ta->Reg-ls
32        |     A1-ms->Td       Reg-ms->Ta
32        +---- Ta->A0-ms       Tb->Reg-ms

Ucode Space:    Common routine: 325
                F+:              13
                F-:              13
                F*:              13
                F/:              25

Performance:    NO-VPs          VPs
                325 cycles      197 cycles per VP
                               +128 cycles for overhead
```

F/: additional 448 cycles

Exception Detection Modes:
Please refer to the section on exception handling at the end of this document. The cost for each mode according to every function is described there.

Instructions: F+,F-,F*,F/
Precision: single
Addr-Mode: two
Condition: cond/always

Algorithm on WTL-3164:



Ucode Space: Common routine: 291
F+: 13
F-: 13
F*: 13
F/: 25

Performance: NO-VPs VPs
165 cycles 100 cycles per VP
+ 65 cycles for overhead

F/: additional 192 cycles

Exception Detection Modes:
Please refer to the section on exception handling at the end of this document. The cost for each mode according to every function is described there.

```
Instructions:   F+,F-,F*
Precision:      single
Addr-Mode:      three
Condition:      cond

Algorithm on WTL-3132:

A op B -> C     32 bit :conditional
#cycles         MB              FB
-------         ----------      ----------
32              B0->Ta          free
32              A0->Tb          Ta->Reg
32        +---->C0->Ta          Tb op Reg->Reg
32        |     B1->Td          Reg->Tc or Ta->Tc
32        |     Tc->C0          Td->Reg
32        +---- A1->Tb          free

Ucode Space:    Common routine: 200
                F+:              13
                F-:              13
                F*:              13

Performance:    NO-VPs          VPs
                165 cycles      132 cycles per VP
                                + 65 cycles for overhead
Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.

*****************************************************************************

Instructions:   F+,F-,F*
Precision:      double
Addr-Mode:      three
Condition:      cond

Algorithm on WTL-3132:
A op B -> C     64 bit :conditional
#cycles         MB              FB
-------         ----------      ----------
Ucode Space:    Common routine:
                F+:
                F-:
                F*:
Performance:    NO-VPs          VPs
Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.
```

```
Instructions:   F+,F-,F*,F/
Precision:      double
Addr-Mode:      three
Condition:      cond

Algorithm on WTL-3164:

A op B -> C     64 :conditional
#cycles         MB              FB
------          ----------      ----------
32              B0-ls->Ta       free
32              B0-ms->Tb       Ta->Reg-ls
32              A0-ls->Tc       Tb->Reg-ms
32        +---->A0-ms->Td       free
32        |     C0-ls->Ta       Tc,Td op Reg->Reg
32        |     C0-ms->Tb       Tc,Td op Reg->Reg
32        |     B1-ls->Td       Reg-ls->Tc or Ta->Tc
32        |     Tc->C0-ls       Td->Reg-ls
32        |     B1-ms->Ta       Reg-ms->Td or Tb->Td
32        |     Td->C0-ms       Ta->Reg-ms
32        +---->A1-ls->Tc       free

Ucode Space:    Common routine: 353
                F+:              13
                F-:              13
                F*:              13
                F/:              25

Performance:    NO-VPs          VPs
                325 cycles      260 cycles per VP
                               + 65 cycles for overhead

F/:             additional 448 cycles

Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.

*************************************************************************

Instructions:   F+,F-,F*,F/
Precision:      single
Addr-Mode:      three
Condition:      cond

Algorithm on WTL-3164:

A op B -> C     32 bit :conditional
#cycles         MB              FB
------          ----------      ----------
32              B0->Ta          free
32              A0->Tb          Ta->Reg
32        +---->C0->Ta          Tb op Reg->Reg
32        |     B1->Td          Reg->Tc or Ta->Tc
32        |     Tc->C0          Td->Reg
32        +---- A1->Tb          free

Ucode Space:    Common routine: 200
                F+:              13
                F-:              13
                F*:              13
                F/:              25

Performance:    NO-VPs          VPs
                165 cycles      132 cycles per VP
                               + 65 cycles for overhead

F/:             additional 192 cycles

Exception Detection Modes:
```

Please refer to the section on exception handling at the end of this document. The cost for each mode according to every function is described there.

```
Instructions:   F+,F-,F*
Precision:      single
Addr-Mode:      three
Condition:      always

Algorithm on WTL-3132:

A op B -> C     32 bit :always
#cycles         MB              FB
-------         ----------      ----------
32              B0->Td          free
32              A0->Ta          Td->Reg
32        +---->B1->Td          Ta op Reg->Reg
32        |     A1->Tb          Reg->Tc or Ta->Tc
32        |     Tc->C0          Td->Reg
32        |     B2->Tc          Tb op Reg->Reg
32        |     A2->Ta          Reg->Td or Tb->Td
32        +-----Td->C1          Tc->Reg

Ucode Space:    Common routine: 291
                F+:              13
                F-:              13
                F*:              13

Performance:    NO-VPs          VPs
                165 cycles      100 cycles per VP
                               + 65 cycles for overhead
Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.

*****************************************************************************

Instructions:   F+,F-,F*
Precision:      double
Addr-Mode:      three
Condition:      always

Algorithm on WTL-3132:
A op B -> C     64 bit :always
#cycles         MB              FB
-------         ----------      ----------
Ucode Space:    Common routine:
                F+:
                F-:
                F*:
Performance:    NO-VPs          VPs
Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.
```

```
Instructions:   F+,F-,F*,F/
Precision:      double
Addr-Mode:      three
Condition:      always

Algorithm on WTL-3164:

A op B -> C     64 :always
#cycles         MB              FB
--------        ---  ------     ----------
32              B0-ls->Ta       free
32              B0-ms->Tb       Ta->Reg-ls
32              A0-ls->Tc       Tb->Reg-ms
32              A0-ms->Td       free
32        +---->B1-ls->Ta       Tc,Td op Reg->Reg
32        |     B1-ms->Tb       Tc,Td op Reg->Reg
32        |     A1-ls->Tc       Reg-ls->Td
32        |     Td->C0-ls       Ta->Reg-ls
32        |     A1-ms->Td       Reg-ms->Ta
32        +---- Ta->C0-ms       Tb->Reg-ms

Ucode Space:    Common routine: 325
                F+:              13
                F-:              13
                F*:              13
                F/:              25

Performance:    NO-VPs          VPs
                325 cycles      197 cycles per VP
                               +128 cycles for overhead

F/:             additional 448 cycles

Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.

*************************************************************************

Instructions:   F+,F-,F*,F/
Precision:      single
Addr-Mode:      three
Condition:      always

Algorithm on WTL-3164:

A op B -> C     32 bit :always
#cycles         MB              FB
--------        ----------      ----------
32              B0->Td          free
32              A0 >Ta          Td->Reg
32        +---->B1->Td          Ta op Reg->Reg
32        |     A1->Tb          Reg->Tc or Ta->Tc
32        |     Tc->C0          Td->Reg
32        |     B2->Tc          Tb op Reg->Reg
32        |     A2->Ta          Reg->Td or Tb->Td
32        +-----Td->C1          Tc->Reg

Ucode Space:    Common routine: 291
                F+:              13
                F-:              13
                F*:              13
                F/:              25

Performance:    NO-VPs          VPs
                165 cycles      100 cycles per VP
                               + 65 cycles for overhead

F/:             additional 192 cycles
```

Exception Detect on Modes:
Please refer to the section on exception handling at the end of this document. The cost for each mode according to every function is described there.

```
Instructions:   F+constant,F-constant,F*constant
Precision:      single
Addr-Mode:      one
Condition:      cond/always

Algorithm on WTL-3132:

A op Const-> A  32 bit :conditional or :always
#cycles         MB              FB
--------        ----------      ----------
2               Const->Bypass   free
1               free            Bypass->Reg[31]
32              A0->Ta          free
32              free            Ta op Reg[31]->Reg
32        +---->A1->Tb          Reg->Tc or Ta->Tc
1         |     free            Bypass->Reg[31]
32        |     Tc->A0          Tb op Reg[31]->Reg
32        |     A2->Ta          Reg->Td or Tb->Td
1         |     free            Bypass->Reg[31]
32        +---- Td->A1          Ta op Reg[31]->Reg

Ucode Space:    Common routine: 200
                F+constant:      13
                F-constant:      13
                F*constant:      13

Performance:    NO-VPs          VPs
                135 cycles       67 cycles per VP
                                +68 cycles for overhead
Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.

*************************************************************************

Instructions:   F+constant,F-constant,F*constant
Precision:      double
Addr-Mode:      one
Condition:      cond/always
Algorithm on WTL-3132:
A op Const-> A  64 bit :always
#cycles         MB              FB
--------        ----------      ----------
Ucode Space:    Common routine:
                F+constant:
                F-constant:
                F*constant:
Performance:    NO-VPs          VPs
Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.
```

```
Instructions:   F+constant,F-constant,F*constant,F/constant
Precision:      double
Addr-Mode:      one
Condition:      cond

Algorithm on WTL-3164:

A op Const-> A  64 bit :conditional
#cycles         MB              FB
-------         ----------      ----------
32              A0-ls->Ta       free
32              A0-ms->Tb       Ta->Reg-ls
32              free            Tb->Reg-ms
5          +--->     Const->Xreg
32         |    free            Reg op Xreg->Reg
32         |    A1-ls->Td       Reg-ls->Tc or Ta->Tc
32         |    Tc->A0-ls       Td->Reg-ls
32         |    A1-ms->Tc       Reg-ms->Td or Tc->Td
32         |    Td->A0-ms       Tc->Reg-ms
5          |         Const->Xreg
32         |    free            Reg op Xreg->Reg
32         |    A2-ls->Ta       Reg-ls->Tb or Td->Tb
32         |    Tb->A0-ls       Ta->Reg-ls
32         |    A2-ms->Tb       Reg-ms->Ta or Tc->Ta
32         +----Ta->A0-ms       Tb->Reg-ms


Ucode Space:    Common routine: 420
                F+constant:      15
                F-constant:      15
                F*constant:      15
                F/constant:      25

Performance:    NO-VPs          VPs
                265 cycles      169 cycles per VP
                               + 96 cycles for overhead

F/:             additional 448 cycles

Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.

*************************************************************************

Instructions:   F+constant,F-constant,F*constant,F/constant
Precision:      double
Addr-Mode:      one
Condition:      always

Algorithm on WTL-3164:

A op Const-> A  64 bit :always
#cycles         MB              FB
-------         ----------      ----------
32              A0-ls->Ta       free
32              A0-ms->Tb       Ta->Reg-ls
32              free            Tb->Reg-ms
5                  Const->Bypass
32              A1-ls->Tc       Reg op Xreg->Reg
32         +---->A1-ms->Td      Reg-ls->Ta
32         |    Ta->A0-ls       Tc->Reg-ls
32         |    A2-ls->Tc       Reg-ms->Tb
32         |    Tb->A0-ms       Ta->Reg-ms
5          +----   Const->Bypass

Ucode Space:    Common routine: 270
                F+constant:      13
                F-constant:      13
                 *constant:      13
```

```
                F/constant:      25

Performance:    NO-VPs           VPs
                265 cycles       128 cycles per VP
                                +133 cycles for overhead

F/:             additional 448 cycles

Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.
```

****

```
Instructions:   F+constant,F-constant,F*constant,F/constant
Precision:      single
Addr-Mode:      one
Condition:      cond/always

Algorithm on WTL-3164:

A op Const-> A  32 bit :conditional or :always
#cycles         MB               FB
-------         ----------       ----------
2               Const->Bypass    free
1               free             Bypass->Reg[31]
32              A0->Ta           free
32              free             Ta op Reg[31]->Reg
32        +---->A1->Tb           Reg->Tc or Ta->Tc
1         |     free             Bypass->Reg[31]
32        |     Tc->A0           Tb op Reg[31]->Reg
32        |     A2->Ta           Reg->Td or Tb->Td
1         |     free             Bypass->Reg[31]
32        +---- Td->A1           Ta op Reg[31]->Reg

Ucode Space:    Common routine: 200
                F+constant:      13
                F-constant:      13
                F*constant:      13
                F/constant:      25

Performance:    NO-VPs           VPs
                135 cycles        67 cycles per VP
                                + 68 cycles for overhead

F/:             additional 192 cycles

Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.
```

```
Instructions:   F+constant,F-constant,F*constant
Precision:      single
Addr-Mode:      two
Condition:      always

Algorithm on WTL-3132:

A op Const-> B  32 bit :always
#cycles         MB              FB
-------         ----------      ----------
2               Const->Bypass   free
1               free            Bypass->Reg[31]
32              A0->Ta          free
32              free            Ta op Reg[31]->Reg
32        +---->A1->Tb          Reg->Tc or Ta->Tc
1         |     free            Bypass->Reg[31]
32        |     Tc->B0          Tb op Reg[31]->Reg
32        |     A2->Ta          Reg->Td or Tb->Td
1         |     free            Bypass->Reg[31]
32        +---- Td->B2          Ta op Reg[31]->Reg

Ucode Space:    Common routine: 200
                F+:              13
                F-:              13
                F*:              13

Performance:    NO-VPs          VPs
                135 cycles       67 cycles per VP
                                +68 cycles for overhead
Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.

*************************************************************************

Instructions:   F+constant,F-constant,F*constant
Precision:      double
Addr-Mode:      two
Condition:      always

Algorithm on WTL-3132:

A op Const-> B  64 bit :always
#cycles         MB              FB
-------         ----------      ----------
Ucode Space:    Common routine:
                F+constant:
                F-constant:
                F*constant:

Performance:    NO-VPs          VPs
Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.
```

```
Instructions:   F+constant,F-constant,F*constant,F/constant
Precision:      double
Addr-Mode:      two
Condition:      always

Algorithm on WTL-3164:

A op Const-> B  64 bit :always
#cycles         MB              FB
--------        ----------      ----------
32              A0-ls->Ta       free
32              A0-ms->Tb       Ta->Reg-ls
32              free            Tb->Reg-ms
5                  Const->Bypass
32              A1-ls->Tc       Reg op Xreg->Reg
32        +---->A1-ms->Td       Reg-ls->Ta
32        |     Ta->B0-ls       Tc->Reg-ls
32        |     A2-ls->Tc       Reg-ms->Tb
32        |     Tb->B0-ms       Ta->Reg-ms
5         +-----   Const->Bypass

Ucode Space:    Common routine: 270
                F+constant:      13
                F-constant:      13
                F*constant:      13
                F/constant:      25

Performance:    NO-VPs          VPs
                265 cycles       128 cycles per VP
                                +133 cycles for overhead

F/:             additional 448 cycles

Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.


*************************************************************************

Instructions:   F+constant,F-constant,F*constant,F/constant
Precision:      single
Addr-Mode:      two
Condition:      always

Algorithm on WTL-3164:

A op Const-> B  32 bit :always
#cycles         MB              FB
--------        ----------      ----------
2               Const->Bypass   free
1               free            Bypass->Reg[31]
32              A0->Ta          free
32              free            Ta op Reg[31]->Reg
32        +---->A1->Tb          Reg->Tc or Ta->Tc
1         |     free            Bypass->Reg[31]
32        |     Tc->B0          Tb op Reg[31]->Reg
32        |     A2->Ta          Reg->Td or Tb->Td
1         |     free            Bypass->Reg[31]
32        +---- Td->B2          Ta op Reg[31]->Reg

Ucode Space:    Common routine: 200
                F+constant:      13
                F-constant:      13
                F*constant:      13
                F/constant:      25

Performance:    NO-VPs          VPs
                135 cycles       67 cycles per VP
                                + 68 cycles for overhead
```

F/: additional 192 cycles

Exception Detection Modes:
Please refer to the section on exception handling at the end of this document. The cost for each mode according to every function is described there.

```
Instructions:   F+constant,F-constant,F*constant
Precision:      single
Addr-Mode:      two
Condition:      cond

Algorithm on WTL-3132:

A op Const-> B  32 bit :conditional
#cycles         MB              FB
-------         ----------      ----------
2               Const->Bypass   free
1               free            Bypass->Reg[31]
32              A0->Ta          free
32        +---->B0->Tb          Ta op Reg[31]->Reg
32        |     A1->Ta          Reg->Td or Tb->Td
32        |     Td->B0          free
1         +-----free            Bypass->Reg[31]


Ucode Space:    Common routine: 135
                F+constant:      13
                F-constant:      13
                F*constant:      13

Performance:    NO-VPs          VPs
                135 cycles      100 cycles per VP
                               + 35 cycles for overhead
Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.

*****************************************************************************

Instructions:   F+constant,F-constant,F*constant
Precision:      double
Addr-Mode:      two
Condition:      cond
Algorithm on WTL-3132:
A op Const-> B  64 bit :conditional
#cycles         MB              FB
-------         ----------      ----------
Ucode Space:    Common routine:
                F+constant:
                F-constant:
                F*constant:
                F/constant:
Performance:    NO-VPs          VPs
Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.
```

```
Instructions:   F+constant,F-constant,F*constant,F/constant
Precision:      double
Addr-Mode:      two
Condition:      cond

Algorithm on WTL-3164:

A op Const-> B  64 bit :conditional
#cycles         MB              FB
--------        ----------      ----------
32              A0-ls->Ta       free
32              A0-ms->Tb       Ta->Reg-ls
32              B0-ls->Ta       Tb->Reg-ms
5         +---->  Const->Bypass
32        |     B0-ms->Tb       Reg op Xreg->Reg
32        |     A1-ls->Td       Reg-ls->Tc or Ta->Tc
32        |     Tc->B0-ls       Td->Reg-ls
32        |     A1-ms->Ta       Reg-ms->Td or Tb->Td
32        |     Td->B0-ms       Ta->Reg-ms
32        +-----B1-ls->Ta       free


Ucode Space:    Common routine: 310
                F+constant:      15
                F-constant:      15
                F*constant:      15
                F/constant:      25

Performance:    NO-VPs          VPs
                265 cycles      197 cycles per VP
                               + 96 cycles for overhead

F/:             additional 448 cycles

Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.

*************************************************************************

Instructions:   F+constant,F-constant,F*constant,F/constant
Precision:      single
Addr-Mode:      two
Condition:      cond

Algorithm on WTL-3164:

A op Const-> B  32 bit :conditional
#cycles         MB              FB
--------        ----------      ----------
2               Const->Bypass   free
1               free            Bypass->Reg[31]
32              A0->Ta          free
32        +----->B0->Tb         Ta op Reg[31]->Reg
32        |     A1->Ta          Reg->Td or Tb->Td
32        |     Td->B0          free
1         +-----free            Bypass->Reg[31]


Ucode Space:    Common routine: 135
                F+constant:      13
                F-constant:      13
                F*constant:      13
                F/constant:      25

Performance:    NO-VPs          VPs
                135 cycles      100 cycles per VP
                               + 35 cycles for overhead
```

F/: additional 192 cycles

Exception Detection Modes:
Please refer to the section on exception handling at the end of this document. The cost for each mode according to every function is described there.

```
Instructions:   F=, F(not =), F>, F(> and =), F<, F(< and =)
Precision:      single
Addr-Mode:      two
Condition:      cond/always

Algorithm on WTL-3132:

A compare B     32 bit :conditional or :always
#cycles         MB                FB
-------         ----------        ----------
32              A0->Ta            free
32          +--->B0->Tb           Ta->Reg
32          |   A1->Tc            Tb compare Reg  <- compare status bits go to Sprint
5           |   status->mem       free            <- producing correct test-flag
32          |   B2->Td            Tc->Reg
32          |   A2->Ta            Td compare Reg  <- compare status bits go to Sprint
5           +---- status->mem     free            <- producing correct test-flag

Ucode Space:    Common routine: 168
                F=:              15
                F:               15
                F>:              15
                F:               15
                F<:              15
                F:               15

Performance:    NO-VPs          VPs
                105 cycles       70 cycles per VP
                                +35 cycles for overhead
Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.

*************************************************************************

The 64 bit compares on WTL-32 will not be supported it will be
faster to do it on the CM using current code.
```

```
Instructions:   F=, F(not =), F>, F(> and =), F<, F(< and =)
Precision:      double
Addr-Mode:      two
Condition:      cond/always

Algorithm on WTL-3164:

A compare B     64 bit :conditional or :always
#cycles         MB              FB
-------         ----------      ----------
32              A0-ls->Ta       free
32              A0-ms->Tb       Ta->Reg-ls
32              B0-ls->Td       free
32        +---->B0-ms->Tc       Tb->Reg-ms
32        |     A1-ls->Ta       Tc,Td compare Reg<- compare status bits go to Sprint
32        |     A1-ms->Tb       Tc,Td compare Reg<- compare status bits go to Sprint
5         |     status->mem     free            <- producing correct test-flag
32        +-----B1-ls->Td       Ta->Reg-ls

Ucode Space:    Common routine: 230
                F=:              15
                F:               15
                F>:              15
                F:               15
                F<:              15
                F:               15

Performance:    NO-VPs          VPs
                200 cycles      135 cycles per VP
                               + 98 cycles for overhead
Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.

*****************************************************************************

Instructions:   F=, F(not =), F>, F(> and =), F<, F(< and =)
Precision:      single
Addr-Mode:      two
Condition:      cond/always

Algorithm on WTL-3164:

A compare B     32 bit :conditional or :always
#cycles         MB              FB
-------         ----------      ----------
32              A0->Ta          free
32        +---->B0->Tb          Ta->Reg
32        |     A1->Tc          Tb compare Reg  <- compare status bits go to Sprint
5         |     status->mem     free            <- producing correct test-flag
32        |     B2->Td          Tc->Reg
32        |     A2->Ta          Td compare Reg  <- compare status bits go to Sprint
5         +-----status->mem     free            <- producing correct test-flag

Ucode Space:    Common routine: 168
                F=:              15
                F:               15
                F>:              15
                F:               15
                F<:              15
                F:               15

Performance:    NO-VPs          VPs
                105 cycles       70 cycles per VP
                               + 35 cycles for overhead
Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.
```

```
Instructions:   Fmin, Fmax
Precision:      single
Addr-Mode:      two
Condition:      cond

Algorithm on WTL-3132:

A max B ->A     32 bit :conditional
#cycles         MB              FB
-------         ----------      ----------
32              A0->Ta          free
32        +---->B0->Tb          Ta->Reg
32        |     A1->Tc          Tb compare Reg  <- compare status bits go to Sprint
5         |     status->mem     free            <- producing correct test-flag
96        |     CM moving max of A and B into A
32        |     B2->Td          Tc->Reg
32        |     A2->Ta          Td compare Reg  <- compare status bits go to Sprint
5         |     status->mem     free            <- producing correct test-flag
96        +-----CM moving max of A and B into A

Ucode Space:    Common routine: 168
                Fmax:            25
                Fmin:            25

Performance:    NO-VPs          VPs
                201 cycles      166 cycles per VP
                               + 35 cycles for overhead
Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.

*************************************************************************

The 64 bit compares on WTL-32 will not be supported it will be
faster to do it on the CM using current code.
```

```
Instructions:   Fmin, Fmax
Precision:      double
Addr-Mode:      two
Condition:      cond

Algorithm on WTL-3164:

A max B ->A     64 bit :conditional or :always
#cycles         MB              FB
--------        ------------    ------------
32              A0-ls->Ta       free
32              A0-ms->Tb       Ta->Reg-ls
32              B0-ls->Td       free
32        +---->B0-ms->Tc       Tb->Reg-ms
32        |     A1-ls->Ta       Tc,Td compare Reg<- compare status bits go to Sprint
32        |     A1-ms->Tb       Tc,Td compare Reg<- compare status bits go to Sprint
5         |     status->mem     free          <- producing correct test-flag
192       |     CM moving max of A and B into A
32        +-----B1-ls->Td       Ta->Reg-ls

Ucode Space:    Common routine: 230
                Fmax:            25
                Fmin:            25

Performance:    NO-VPs          VPs
                392 cycles      327 cycles per VP
                               + 98 cycles for overhead
Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.

************************************************** ***************

Instructions:   Fmin, Fmax
Precision:      single
Addr-Mode:      two
Condition:      cond

Algorithm on WTL-3164:

A max B ->A     32 bit :conditional
#cycles         MB              FB
--------        ------------    ------------
32              A0->Ta          free
32        +---->B0->Tb          Ta->Reg
32        |     A1->Tc          Tb compare Reg  <- compare status bits go to Sprint
5         |     status->mem     free            <- producing correct test-flag
96        |     CM moving max of A and B into A
32        |     B2->Td          Tc->Reg
32        |     A2->Ta          Td compare Reg  <- compare status bits go to Sprint
5         |     status->mem     free            <- producing correct test-flag
96        +-----CM moving max of A and B into A

Ucode Space:    Common routine: 168
                Fmax:            25
                Fmin:            25

Performance:    NO-VPs          VPs
                201 cycles      166 cycles per VP
                               + 35 cycles for overhead
Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.
```

```
Instructions:   Fmin, Fmax
Precision:      single
Addr-Mode:      three
Condition:      cond

Algorithm on WTL-3132:

A max B ->C     32 bit :conditional
#cycles         MB              FB
--------        ----------      ----------
32              A0->Ta          free
32        +---->B0->Tb          Ta->Reg
32        |     A1->Tc          Tb compare Reg  <- compare status bits go to Sprint
5         |     status->mem     free            <- producing correct test-flag
128       |     CM moving max of A and B into C
32        |     B2->Td          Tc->Reg
32        |     A2->Ta          Td compare Reg  <- compare status bits go to Sprint
5         |     status->mem     free            <- producing correct test-flag
128       +-----CM moving max of A and B into C

Ucode Space:    Common routine: 168
                Fmax:            30
                Fmin:            30

Performance:    NO-VPs          VPs
                233 cycles      198 cycles per VP
                               + 35 cycles for overhead
Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.

*************************************************************************

The 64 bit compares on WTL-32 will not be supported it will be
faster to do it on the CM using current code.
```

```
 nstructions:   Fmin, Fmax
Precision:      double
Addr-Mode:      three
Condition:      cond

Algorithm on WTL-3164:

A max B ->C     64 bit :conditional or :always
#cycles         MB              FB
-------         ----------      ---------
32              A0-ls->Ta       free
32              A0-ms->Tb       Ta->Reg-ls
32              B0-ls->Td       free
32        +---->B0-ms->Tc       Tb->Reg-ms
32        |     A1-ls->Ta       Tc,Td compare Reg<- compare status bits go to Sprint
32        |     A1-ms->Tb       Tc,Td compare Reg<- compare status bits go to Sprint
5         |     status->mem     free            <- producing correct test-flag
256       |     CM moving max of A and B into C
32        +-----B1-ls->Td       Ta->Reg-ls

Ucode Space:    Common routine: 230
                F=:              15
                F:               15
                F>:              15
                F:               15
                F<:              15
                F:               15

Performance:    NO-VPs          VPs
                456 cycles      391 cycles per VP
                               + 98 cycles for overhead
Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.

*************************************************************************

Instructions:   Fmin, Fmax
Precision:      single
Addr-Mode:      three
Condition:      cond

Algorithm on WTL-3164:

A max B ->C     32 bit :conditional

#cycles         MB              FB
-------         ----------      ----------
32              A0->Ta          free
32        +---->B0->Tb          Ta->Reg
32        |     A1->Tc          Tb compare Reg  <- compare status bits go to Sprint
5         |     status->mem     free            <- producing correct test-flag
128       |     CM moving max of A and B into C
32        |     B2->Td          Tc->Reg
32        |     A2->Ta          Td compare Reg  <- compare status bits go to Sprint
5         |     status->mem     free            <- producing correct test-flag
128       +-----CM moving max of A and B into C

Ucode Space:    Common routine: 168
                Fmax:            30
                Fmin:            30

Performance:    NO-VPs          VPs
                233 cycles      198 cycles per VP
                               + 35 cycles for overhead
Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.
```

```
Instructions:   Fcos, Fsin, Ftan, (↑ e x), (log x) and etc..
Precision:      single
Addr-Mode:      one
Condition:      cond/always

Algorithm on WTL-3132:

Third order polynomial evaluation for all of these instructions.

Poly (A) ->A    32 bit  :conditional or :always

Comment:  Constants for various functions have already been
loaded into the Weitek chip.  They are in temp registers

#cycles         MB              FB
-------         ----------      ----------
32              A0->Ta          free
32              A1->Tb          (Ta * C3)  + C2->Reg
32        +---->free            (Ta * Reg) + C1->Reg
32        |     free            (Ta * Reg) + C0->Reg
32        |     free            Reg ->Tc
32        |     Tc ->A0         (Tb * C3)  + C2->Reg
32        |     A2->Ta          (Tb * Reg) + C1->Reg
32        |     free            (Tb * Reg) + C0->Reg
32        |     free            Reg ->Td
32        +-----Td ->A0         (Ta * C3)  + C2->Reg


Ucode Space:    Common routine: 330
                Fcos:            20
                Fsin:            20
                Ftan:            20
                (↑ e x):         20
                (log x):         20

Performance:    NO-VPs          VPs
                200 cycles      135 cycles per VP
                               + 65 cycles for overhead
Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.

*************************************************************************

Instructions:   Fcos, Fsin, Ftan, (↑ e x), (log x) and etc..
Precision:      double
Addr-Mode:      one
Condition:      cond/always

Algorithm on WTL-3132:

Third order polynomial evaluation for all of these instructions.

Poly (A) ->A    32 bit  :conditional or :always

Comment:  This code will have to be implemented in Lisp
as a subroutine calling the double precision multiply
and double precision add code.

Performance:    NO-VPs          VPs
                (+ (* 3 F*)     (+ (* 3 F*)
                   (* 3 F+))       (* 3 F+)

Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.
```

```
Instructions:   Fcos, Fsin, Ftan, (↑ e x), (log x) and etc..
Precision:      double
Addr-Mode:      one
Condition:      cond/always

Algorithm on WTL-3164:

Third order polynomial evaluation for all of these instructions.

Poly (A) ->A    64 bit  :conditional or :always

Comment:  Constants for various functions have already been
loaded into the Weitek chip.  They are in temp registers

#cycles         MB              FB
-------         ----------      ----------
32          +---->A0-ls->Ta     free
32          |   A0-ms->Tb       free
64          |   free            (Ta,Tb * C3)  + C2->Reg
64          |   free            (Ta,Tb * Reg) + C1->Reg
64          |   free            (Ta,Tb * Reg) + C0->Reg
32          |   free            Reg-ls->Tc or Ta->Tc
32          |   Tc->A0-ls       Reg-ms->Td or Tb->Td
32          +-----Td->A0-ms     free


Ucode Space:    Common routine: 400
                Fcos:            20
                Fsin:            20
                Ftan:            20
                (↑ e x):         20
                (log x):         20

Performance:    NO-VPs          VPs
                360 cycles      360 cycles

Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.

*************************************************************************

Instructions:   Fcos, Fsin, Ftan, (↑ e x), (log x) and etc..
Precision:      single
Addr-Mode:      one
Condition:      cond/always

Algorithm on WTL-3164:

Third order polynomial evaluation for all of these instructions.

Poly (A) ->A    32 bit  :conditional or :always

Comment:  Constants for various functions have already been
loaded into the Weitek chip.  They are in temp registers

#cycles         MB              FB
-------         ----------      ----------
32              A0->Ta          free
32              A1->Tb          (Ta * C3)  + C2->Reg
32          +---->free          (Ta * Reg) + C1->Reg
32          |   free            (Ta * Reg) + C0->Reg
32          |   free            Reg ->Tc
32          |   Tc ->A0         (Tb * C3)  + C2->Reg
32          |   A2->Ta          (Tb * Reg) + C1->Reg
32          |   free            (Tb * Reg) + C0->Reg
32          |   free            Reg ->Td
32          +-----Td ->A0       (Ta * C3)  + C2->Reg
```

```
Ucode Space:    Common routine: 330
                Fcos:            20
                Fsin:            20
                Ftan:            20
                (↑ e x):         20
                (log x):         20

Performance:    NO-VPs          VPs
                200 cycles       135 cycles per VP
                                +  65 cycles for overhead
Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.
```

```
Instructions:   Global (F+, F-, F*)
Precision:      single
Addr-Mode:      one
Condition:      cond/always

Algorithm on WTL-3132:

The first 32 values get collapsed on every chip.
Dimension zero...

Comment:  This code uses cube swaps to achieve the
global value

#cycles         MB              FB
-------         --              --
32              A0->Ta          free
32              free            Ta->Reg
33              free            Reg op Reg->Reg[1]
1               free            Reg[0]->Ta[1]
1               Ta[1]->MEM      free
(* 2 32)        cube swap along dimension 1
32              MEM->Ta[1]      free
1               free            Ta[1] op Reg[1]->Reg[2]
1               free            Reg->Ta[2]
1               Ta[2]->MEM      free
(* 2 32)        cube swap along dimension 2
32              MEM->Ta[2]      free
1               free            Ta[2] op Reg[2]->Reg[3]
1               free            Reg->Ta[3]
1               Ta[3]->MEM      free
(* 2 32)        cube swap along dimension 3
.               .               .
.               .               .
.               .               .

Ucode Space:    Common routine: 450
                Global F+:       30
                Global F-:       30
                Global F*:       30


Performance:    NO-VPs          VPs
                1100 cycles     1000 cycles per VP
                               +  96 cycles for overhead
Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.

*************************************************************************
```

```
Instructions:   Global (F+, F-, F*)
Precision:      double
Addr-Mode:      one
Condition:      cond/always

Algorithm on WTL-3164:

The first 64 values get collapsed on every chip.
Dimension zero...

Comment:  This code uses cube swaps to achieve the
global value.  It will also need for conditional
global multiply to have zero source on the FB port.

#cycles         MB              FB
-------         ----------      ----------
32              A0-ls->Ta       free
32              A0-ms->Tb       Ta->Reg-ls
32              free            Tb->Reg-ms
33              free            Reg op Reg->Reg[1]
2               free            Reg[1]->Ta[1],Tb[1]
2               Ta[1],Tb[1]->mem free
(* 2 64)        cube swap dimension 1
64              MEM->Ta[1],Tb[1]free
2               free            Ta[1] op Reg[1]->Reg[2]
2               free            Reg[2]->Ta[2],Tb[2]
2               Ta[2],Tb[2]->MEMfree
(* 2 64)        cube swap dimension 2
.               .               .
.               .               .
.               .               .

Ucode Space:    Common routine: 450
                Global F+:       30
                Global F-:       30
                Global F*:       30


Performance:    NO-VPs          VPs
                2200 cycles     2000 cycles per VP
                              +  96 cycles for overhead
Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.

*************************************************************************

Instructions:   Global (F+, F-, F*)
Precision:      single
Addr-Mode:      one
Condition:      cond/always

Algorithm on WTL-3164:

The first 32 values get collapsed on every chip.
Dimension zero...

Comment:  This code uses cube swaps to achieve the
global value

#cycles         MB              FB
-------         ----------      ----------
32              A0->Ta          free
32              free            Ta->Reg
33              free            Reg op Reg->Reg[1]
1               free            Reg[1]->Ta[1]
1               Ta[1]->MEM      free
(* 2 32)        cube swap along dimension 1
32              MEM->Ta[1]      free
1               free            Ta[1] op Reg[1]->Reg[2]
```

```
1               free            Reg->Ta[2]
1               Ta[2]->MEM      free
(* 2 32)        cube swap along dimension 2
32              MEM->Ta[2]      free
1               free            Ta[2] op Reg[2]->Reg[3]
1               free            Reg->Ta[3]
1               Ta[3]->MEM      free
(* 2 32)        cube swap along dimension 3
.               .               .
.               .               .
.               .               .

Ucode Space:    Common routine: 450
                Global F+:       30
                Global F-:       30
                Global F*:       30


Performance:    NO-VPs          VPs
                1100 cycles     1000 cycles per VP
                                + 96 cycles for overhead
Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.
```

```
Instructions:   F/, F/constant
Precision:      single
Addr-Mode:      one,two,three
Condition:      cond/always

Algorithm on WTL-3132:

Uses Newton-Raphson method to obtain a/b

A divide B -> A 32 bit :conditional or always

register<- seed 1/b
(let* ((first-iteration (* register (- 2 (* source register))))
      (second-iteration (* first-iteration (- 2 (* source first-iteration))))))

The pipelined flowchart for this algorithm is hard to desribe on paper.
For those that are really interested please refer to the actual microcode.

Ucode Space:    Common routine: 291
                F/:              13
                F/constant:      13

Performance:    NO-VPs          VPs
F/(two):        300 cycles       256 cycles per VP
                                +32 cycles for overhead
F/(three):      300 cycles       256 cycles per VP
                                +32 cycles for overhead

F/constant(two): 260 cycles       64 cycles per VP
                                 256 cycles for overhead

F/constant(three): 260 cycles     96 cycles per VP
                                 256 cycles for overhead


Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.

*************************************************************************

Doing a double precision divide on WTL-3132 can be done in two different
ways.  First the way we are doing in now (performance 70 MFLOPS on Beta), or
using the double precision multiply written for the WTL-3132 and using
table-lookup capabilities of the Sprint chip.  This performance will be equal
to the following:

1.  Look-up a 16 bit value from a common table (performance of 16 bit aref)
    A table would have to be 64k bits for this.  A smaller table could be used,
    but that would requeire longer time to compute the result.

2.  do 5 multiplies and 2 subtracts (on double precision data)

Performance: (+ (16 bit aref) + (* 5 F*) + (* 2 F-))
```

```
Instructions:   Dot Product, Outer Product
Precision:      single
Addr-Mode:      three
Condition:      cond/always

Algorithm on WTL-3132:

Comment:
dot <- sum [a(i)*b(i)]
outer <- * [a(i)+b(i)]
a(i) and b(i) reside in the same processor...


A op B -> A     32 bit :conditional or :always
#cycles         MB              FB
-------         ----------      ----------
32              A0->Ta          free
32              B0->Tb          Ta->Reg
32              A1->Ta          Tb op Reg->Reg
33              Bypass->MEM     Reg->Bypass
32        +------>B1->Tb        Ta->Reg
32        |     A2->Ta          Tb op Reg->Reg
33        |     MEM->Bypass     Bypass op Reg->Reg
33        +-------Bypass->MEM   Reg->Bypass
.               .               .
.               .               .
.               .               .
32              free            Reg->Ta
32              Ta->Dot         free

Ucode Space:    Common routine: 350
                Dot Product:     13
                Outer Product    13

Performance:     128 cycles per element for condition or always
                +192 cycles overhead for always
                +224 cycles overhead for conditional

Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.

*************************************************************************

Instructions:   Dot Product, Outer Product
Precision:      double
Addr-Mode:      three
Condition:      cond/always

Algorithm on WTL-3132:

Comment:
dot <- sum [a(i)*b(i)]
outer <- * [a(i)+b(i)]
a(i) and b(i) reside in the same processor...

This will have to be implemted in Lisp as a simple
loop that does the above equation.  Performance is
based on the double precision F* and double precision
F+.
```

```
Instructions:   Dot Product, Outer Product
Precision:      double
Addr-Mode:      three
Condition:      cond/always

Algorithm on WTL-3164:

Comment:
dot <- sum [a(i)*b(i)]
outer <- * [a(i)+b(i)]
a(i) and b(i) reside in the same processor...


A op B -> A     32 bit :conditional or :always
#cycles         MB              FB
-------         ----------      ----------
32              A0-ls->Ta       free
32              A0-ms->Tb       Ta->Reg-ls
32              B0-ls->Ta       Tb->Reg-ms
32              B0-ms->Tb       free
32              A1-ls->Tc       Ta,Tb op Reg->Reg
32              A1-ms->Td       Ta,Tb op Reg->Reg
65        +------>Bypass->Mem   Reg->Bypass
32        |     B1-ls->Ta       Td->Reg-ms
32        |     B1-ms->Tb       Tc->Reg-ms
32        |     A1-ls->Tc       Ta,Tb op Reg->Reg
32        |     A1-ms->Td       Ta,Tb op Reg->Reg
65        +------Mem->Bypass    Bypass op Reg->Reg
.               .               .
.               .               .
                .               .
32              free            Reg-ls->Ta
32              Ta->Dot-ls      Reg-ms->Tb
32              Tb->Dot-ms      free

Ucode Space:    Common routine: 450
                Dot Product:     13
                Outer Product    13

Performance:     258 cycles per element for condition or always
                +288 cycles overhead for always
                +320 cycles overhead for conditional

Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.


*************************************************************************

Instructions:   Dot Product, Outer Product
Precision:      single
Addr-Mode:      three
Condition:      cond/always

Algorithm on WTL-3164:

Comment:
dot <- sum [a(i)*b(i)]
outer <- * [a(i)+b(i)]
a(i) and b(i) reside in the same processor...


A op B -> A     32 bit :conditional or :always
#cycles         MB              FB
-------         ----------      ----------
32              A0->Ta          free
32              B0->Tb          Ta->Reg
32              A1->Ta          Tb op Reg->Reg
33              Bypass->MEM     Reg->Bypass
```

```
32        +------>B1->Tb          Ta->Reg
32        |       A2->Ta          Tb op Reg->Reg
33        |       MEM->Bypass     Bypass op Reg->Reg
33        +-------Bypass->MEM     Reg->Bypass
.                 .               .
.                 .               .
.                 .               .
32                free            Reg->Ta
32                Ta->Dot         free

Ucode Space:      Common routine: 350
                  Dot Product:     13
                  Outer Product    13

Performance:       128 cycles per element for condition or always
                  +192 cycles overhead for always
                  +224 cycles overhead for conditional
Exception Detection Modes:
Please refer to the section on exception handling at the
end of this document.  The cost for each mode according
to every function is described there.
```

Exception Modes:

There are 4 exception modes:

1. Record certain (overflow) fatal errors
2. Trap on certain (overflow) fatal errors
3. Record any IEEE exception
4. Trap on any IEEE exception

1. Record Fatal Errors:

We will support few fatal errors:

OVERFLOW
DIVIDE BY ZERO

The implementation is as follows.

Algorithm for Fatal Error Recording on WTL-3132:

The chip provides the capabilities to detect overflow
and we will use that.

OVERFLOW:

The overflow detection is enabled and the status of that
comes out on the FPEX pin. The sprint chip will cache that
bit into its status transposer and at the end of the VP execution
the bit in the status transposer will be swapped out and ORed
with the VP bit for the overflow exception.

Time to detect and set VP flag: 4 cycles

This algorithm applies to all instructions with an exception
of polynomial evaluation instructions. They are handled somewhat
differently. After every iteration the bits of the status transposer
are swapped out and ORed with VP flag in memory.

Time to detect and set VP flag: (* number of iterations 4) cycles

DIVIDE BY ZERO:

This needs to be detected only for the divide operation.
The microcode will scan the exponent of the divisor to
see if it's zero. If so then the microcode will set the VP
divide by zero flag and will turn that (those) processor(s)
off and will continue with the execution for other processors.

Time to detect and set Vp flag: 20 cycles

Algorithm for Fatal Error Recording on WTL-3164:

The chip provides the capabilities to detect overflow
and we will use that. In this case we might want to allow
some other set of fatal errors, when this chip is used
in the system since it has a much bigger detection
capabilities.

OVERFLOW:

The overflow detection is enabled and the status of that
comes out on the status pins. The sprint chip will cache that
bit into its status transposer and at the end of the VP execution
the bit in the status transposer will be swapped out and ORed
with the VP bit for the overflow exception.

Time to detect and set VP flag: 4 cycles

This algorithm applies to all instructions with an exception

of polynomial evaluation instructions. They are handled somewhat differently. After every iteration the bits of the status transposer are swapped out and ORed with VP flag in memory.

Time to detect and set VP flag: (* number of iterations 4) cycles

DIVIDE BY ZERO:

This needs to be detected only for the divide operation.

The divide by zero detection is enabled and the status of that comes out on the status pins. The sprint chip will cache that bit into its status transposer and at the end of the VP execution the bit in the status transposer will be swapped out and ORed with the VP bit for the divide by zero exception.

Time to detect and set VP flag: 4 cycles

2. Trap and Record Fatal Errors:

We will support traps on the following fatal errors:

OVERFLOW
DIVIDE BY ZERO

The implementation is as follows.

Algorithm for Trap on Fatal Error and Recording on WTL-3132 and WTL-3164:

This applies to both chips and just to the OVERFLOW on WTL-3132
and to the OVERFLOW and DIVIDE BY ZERO on WTL-3164:

If traps are enabled by the user the following happens.

1. the status transposer is swapped out to memory
   (exception bit is swapped to memory)

2. Then exception bit is is fed through the Global OR by the CM chips

3. If the global asserted we enter the Fatal Errors
   trap handler otherwise go to next VP. If no more VPs
   then we are done.

Trap handler:

The exception bit is in memory.
The result that caused that exception is still in the
Weitek chip.

1. Move the result out of the Weitek Reg-file into
   memory.

2. Select the processors that had an exception
   cm:context-flag <- cm:context-flag and exception-flag

3. Deliver the proper value into their location based
   on the context-flag (cm chip does that).

4. Turn on the processors that had no exception,
   deliver the correct result into their destination.

5. Or the exception bit with VP exception flag.

6. Exit trap handler. Signal to the user about the error.

Time to detect and enter a Trap on Fatal Errors: 6 cycles.

The divide by zero error on WTL-3132 is handled by a diffrent trap
hand'er that does involve the WTL-3132 and is handled totally in
software.

3. Record any IEEE exception:

We will support all five IEEE exceptions with a certain cost associated with each one.

IVALID OPERATION
OVERFLOW
DIVIDE BY ZERO
UNDERFLOW
INEXACT

The implementation is as follows.

Algorithm for Recording IEEE exceptions on WTL-3132:

This applies only to single precision, since double precision will be done in software with some assistance from hadrware and there will be no problems on detecting any exception or all exceptions.

IVALID OPERATION:

Both operands will be scaned first before an operation proceeds if an invalid operation is found a VP flag will be set. This is relatively expensive operation since we might have to do few scan over the numbers to determine the correct invalid operation.

Time to detect this exception: Varies with an operation the best would be one scan per exponent of each operand. Hence the best average time would 36 cycles to set a VP flag.

OVERFLOW:

The overflow detection is enabled and the status of that comes out on the FPEX pin. The sprint chip will cache that bit into its status transposer and at the end of the VP execution the bit in the status transposer will be swapped out and ORed with the VP bit for the overflow exception.

Time to detect and set VP flag: 4 cycles

This algorithm applies to all instructions with an exception of polynomial evaluation instructions. They are handled somewhat differently. After every iteration the bits of the status transposer are swapped out and ORed with VP flag in memory.

Time to detect and set VP flag: (* number of iterations 4) cycles

DIVIDE BY ZERO:

This needs to be detected only for the divide operation. The microcode will scan the exponent of the divisor to see if it's zero. If so then the microcode will set the VP divide by zero flag and will turn that (those) processor(s) off and will continue with execution for other processors.

Time to detect and set Vp flag: 20 cycles

UNDERFLOW:

The underflow detection is enabled and the status of that comes out on the FPEX pin. The sprint chip will cache that bit into its status transposer and at the end of the VP execution the bit in the status transposer will be swapped out and ORed with the VP bit for the underflow exception.

Time to detect and set VP flag: 4 cycles

If both the underflow and overflow are enabled then we can not differentiate which one exactly happend overflow or underflow. Since the chip provides us with an OR of both. One solution could be to give user just that (makes us not fully IEEE). Second re-execute the instruction with one exception turned on and the other off, so that we can correctly determine the proper exception. Third is to do one operation at a time and keep reading the status register every time an instruction executes. This lengthens the operations loop by (* 32 4) cycles for all instructions.

This algorithm applies to all instructions with an exception of polynomial evaluation instructions. They are handled somewhat differently. After every iteration the bits of the status transposer are swapped out and ORed with VP flag in memory.

Time to detect and set VP flag: (* number of iterations 4) cycles

If both overflow and underflow are enabled then the time increases by

(* number of iterations (* 32 4))

INEXACT:

Can not be done using the chip directly. One can uses the chip though just to work on portion of the significand and then uses the software to produce the correct result and the proper inexact exception.

Cost is very significand, it depends on the time to do a variable precision multiply or add using the chip. The times for those are described further on.

Algorithm for Recording IEEE exceptions on WTL-3164:

This applies to both precisions single or double precision.

IVALID OPERATION:

The invalid operation detection is enabled and the status of that
comes out on the status pins. The sprint chip will cache that
bit into its status transposer and at the end of the VP execution
the bit in the status transposer will be swapped out and ORed
with the VP bit for the overflow exception.

Time to detect and set VP flag: 4 cycles

If the design of the Weitek chip does not provide us with the status
for this operation, then the algorithm from WTL-3132 can be applied:

Both operands will be scaned first before an
operation proceeds if an invalid operation is found
a VP flag will be set. This is relatively expensive
operation since we might have to do few scan over the
numbers to determine the correct invalid operation.

Time to detect this exception: Varies with an operation
the best would be one scan per exponent of each operand.
Hence the best average time would 36 cycles to set a VP flag.

OVERFLOW:

The overflow detection is enabled and the status of that
comes out on the status pins. The sprint chip will cache that
bit into its status transposer and at the end of the VP execution
the bit in the status transposer will be swapped out and ORed
with the VP bit for the overflow exception.

Time to detect and set VP flag: 4 cycles

This algorithm applies to all instructions with an exception
of polynomial evaluation instructions. They are handled somewhat
differently. After every iteration the bits of the status transposer
are swapped out and ORed with VP flag in memory.

Time to detect and set VP flag: (* number of iterations 4) cycles

DIVIDE BY ZERO:

The divide by zero detection is enabled and the status of that
comes out on the status pins. The sprint chip will cache that
bit into its status transposer and at the end of the VP execution
the bit in the status transposer will be swapped out and ORed
with the VP bit for the divide by zero exception.

Time to detect and set VP flag: 4 cycles

UNDERFLOW:

The underflow detection is enabled and the status of that
comes out ton he status pins. The sprint chip will cache that
bit into its status transposer and at the end of the VP execution
the bit in the status transposer will be swapped out and ORed
with the VP bit for the underflow exception.

Time to detect and set VP flag: 4 cycles

If the design of the Weitek chip does not provide us with the status
for this operation, then the algorithm from WTL-3132 can be applied:

To do one operation at a time and keep reading the status register every
time an instruction executes. This lengthens the operations loop by (* 32 4)
cycles for single precision operations and by (* 64 4) cycles for double precision

operations.

This algorithm applies to all instructions with an exception of polynomial evaluation instructions. They are handled somewhat differently. After every iteration the bits of the status transposer are swapped out and ORed with VP flag in memory.

Time to detect and set VP flag: (* number of iterations 4) cycles

If the design of the Weitek chip does not provide us with the status for this operation, then the algorithm from WTL-3132 can be applied and the time proportionally increase to:

(* number of iterations (* 32 4)) for single precision
(* number of iterations (* 64 4)) for double precision

INEXACT:

The inexact detection is enabled and the status of that comes out on the status pins. The sprint chip will cache that bit into its status transposer and at the end of the VP execution the bit in the status transposer will be swapped out and ORed with the VP bit for the underflow exception.

Time to detect and set VP flag: 4 cycles

If the design of the Weitek chip does not provide us with the status for this operation, then the following algorithm can be applied:

To do one operation at a time and keep reading the status register every time an instruction executes. This lengthens the operations loop by (* 32 4) cycles for single precision operations and by (* 64 4) cycles for double precision operations.

2. Trap and Record IEEE exceptions:

We will support traps on the all IEEE exceptions:

IVALID OPERATION
OVERFLOW
DIVIDE BY ZERO
UNDERFLOW
INEXACT

The implementation is as follows.

Algorithm for Trap on IEEE exceptions and Recording on WTL-3132 and WTL-3164:

This applies to both chips.

If traps are enabled by the user the following happens.

1. the status transposer is swapped out to memory
   (exception bit is swapped to memory)

2. Then exception bit is is fed through the Global OR by the CM chips

3. If the global asserted we enter the IEEE exceptions
   trap handler otherwise go to next VP. If no more VPs
   then we are done. The trap handler differs from one chip
   to the other. If an exception can be detected in hardware
   then the trap handler with hardware assistance is entered
   otherwise a trap handler for the software assistance is entered.

Trap handler with hardware assistance:

The exception bit is in memory.
The result that caused that exception is still in the
Weitek chip.

1. Move the result out of the Weitek Reg-file into
   memory.

2. Select the processors that had an exception
   cm:context-flag <- cm:context-flag and exception-flag

3. Deliver the proper value into their location based
   on the context-flag (cm chip does that).

4. Turn on the processors that had no exception,
   deliver the correct result into their destination.

5. Or the exception bit with VP exception flag.

6. Exit trap handler. Signal to the user about the error.

Time to detect and enter a Trap on IEEE exception: 6 cycles.

Trap handler with software assistance:

The exception bit is in memory.
The result that caused that exception is still being
manupulated by the software(microcode).

1. Select the processors that had an exception
   cm:context-flag <- cm:context-flag and exception-flag

3. Deliver the proper value into their location based
   on the context-flag (cm chip does that).

4. Turn on the processors that had no exception,
   deliver the correct result into their destination.

5. Or the exception bit with VP exception flag.

6. Exit trap handler. Signal to the user about the error.

Time to detect and enter a Trap on IEEE exception: 6 cycles

Doing double precision math on WTL-3132

The following are considered: F*, F+ and F-, F/

Algorithm for implemeting double precision F* using WTL-3132:

Since the multiplier on the WTL-3132 is only 24 bits wide and the result that we can obtain from it is 23 bits, then we can only multiply 11 bits at a time. The significand 53 bits gets partitioned into chunks of 11 (almost 5 chunks)

We are doing A*B->A

0. some setup time to load upper portion of the word with correct value for multiplication, takes 10 cycles.

1. first load (B) values into :transposer-a the bits from <22:11> are the valid bits that we will multiply together. (11 cycles)

2. load another batch of (A) values into :transposer-b the bits from <22:11> are the valid bits that we will multiply together. (11 cycles)

3. Multiply, takes 32 cycles.

4. unload result into :transposer-c. Also load next 11 other bits of (A) into :transposer-a. (takes 32 cycles)

5. Store first 22 bits of the first partial product into memory. (takes 22 cycles)

6. Do the add the other partial product (* 3 22 cycles)

7. go back to step 3 for 4 more iterations

8. go back to step 1 for 4 more iterations

Time to just multiply the significands: 3750 cycles

Time to add the exponents: 33 cycles

Hence total time to do F* = 3800 cycles or ~140 MFLOPS

This time does not include any overhead for exception handling.

Algorithm for implemeting double precision F+ and F- using WTL-3132:

It is not worth using the WTL-3132 to do F+ and F-. The floating point addition algorithm requiers a shift left or right. Since we can only obtain shift left by using the chip and only for certain length at a time, but we also have to know how many leading zeros the significand had so that the epxonent can be adjusted accordingly. Since we would have to determine that anyway and we can not use the chip to help us with that, we might us well do the shift as we scan for leading zeros. Meaning the shift will be done on the CM.

Time for double precision F+ and F- = 3900 cycles or ~140 MFLOPS

This time does not include any overhead for exception handling.

Algorithm for implemeting double precision F/ using WTL-3132:

We can aplly two algoritms to this problem. One is Newton-Raphson method which is a multiplicative method it requeres with 2 multiplies and one subtract per iteration. The number of iterations depends on how long the initial seed is. One iterations takes 11400 cycles. To do a double precision F/ it will take (* 7 11400) cycles or a total time of:

Time for double precision F+ and F- = 80000 cycles or ~6 MFLOPS

If one uses the current divide algorithm that time takes. This uses the CM chip and not the Weitek.

Time for double precision F+ and F- = 8000 cycles or ~60 MFLOPS

This time does not include any overhead for exception handling.